Today's topics

- The TEI and its architecture
- Working with the schema generator

How does the TEI scheme work? In today's exercise, you'll learn how to build your very own schema.

XML: a licence for ill?

XML allows you to make up your own tags, and doesn't require a DTD...

- The XML concept is dangerously powerful:
  - XML elements are light in semantics
  - one man's <p> is another's <para> (or is it?)
  - the appearance of interchangeability may be worse than its absence
- But XML is still too good to ignore
  - mainstream software development
  - proliferation of tools
  - the future of the web

What kind of grammar do you need?

- To get the best out of XML, you need two kinds of grammar:
  - document type **declaration**: names for your elements, attributes, entities, notations (syntactic constraints)
  - document type **definition**: usage and meaning constraints on the foregoing
- Published specifications usually combine the two, hence they lack modularity

Some answers

- Rolling your own schema
  - ... starting from scratch
  - ... by combining snippets, preferably from an existing conceptual framework (aka architecture)
- Customizing someone else's schema
  - definitions should be meaningful within a given user community
  - declarations should be appropriate to a given set of applications
- The TEI provides a good candidate architecture
  - Namespaces do not provide the whole answer (though at least they remind us the problem exists)

The TEI what?

- Originally, a research project within the humanities
- Sponsored by three professional associations
- Funded 1990-1994 by US NEH, EU LE Programme et al
- Major influences
  - digital libraries and text collections
  - language corpora
  - scholarly datasets
- International consortium established June 1999 (see http://www.tei-c.org/)

| Taming the TEI Tiger | 1 |
| Taming the TEI Tiger | 2 |
| Taming the TEI Tiger | 3 |
| Taming the TEI Tiger | 4 |
| Taming the TEI Tiger | 5 |
| Taming the TEI Tiger | 6 |
Goals of the TEI

- Better interchange and integration of scholarly data
- Support for all texts, in all languages, from all periods
- Guidance for the perplexed: what to encode — hence, a user-driven codification of existing best practice
- Assistance for the specialist: how to encode — hence, a loose framework into which unpredictable extensions can be fitted

These apparently incompatible goals result in a highly flexible, modular environment.

TEI Deliverables

- A set of recommendations for text encoding, covering both generic text structures and some highly specific areas based on (but not limited by) existing practice
- A very large collection of element definitions combined into a very loose document type declaration
- A mechanism for creating multiple views (schemas) of the foregoing
- One such view and associated tutorial: TEI Lite (http://www.tei-c.org/TEI/Lite/)

For the full picture see http://www.tei-c.org/TEI/Guidelines/

Legacy of the TEI

- A way of looking at what “text” really is
- A codification of current scholarly practice
- (Crucially) a set of shared assumptions and priorities about the digital agenda:
  - Focus on content and function (rather than presentation)
  - Identify generic solutions (rather than application-specific ones)

Designing a schema for the TEI

- How can a single mark-up scheme handle a large variety of requirements?
  - All texts are alike
  - Every text is different
- Learn from the database designers
  - One construct, many views
  - Each view a selection from the whole

How many schemas do we need?

- One (the Corporate or WKWBFY approach)
- None (the Anarchic or NWEUMP approach)
- As many as it takes (the Mixed Economy or XML approach)
- Or a single main schema with many faces (a British schema)

Core modules

- Infrastructure module: element classes and macros
- Detailed metadata provision: the TEI Header
- Core module: defines a large set of common textual requirements:
  - Paragraphs
  - Highlighted phrases
  - Names, dates, number, abbreviations...
  - Editorial tags
  - Notes, cross-references, bibliography
- Specialised structure modules for:
  - “Book like” prose, verse, and drama
  - Transcribed speech
  - Dictionaries and lexica
**Additional modules**
- sets of elements for specialised application areas
- can be mixed and matched ad lib
- currently provided:
  - linking and alignment; analysis; feature structures;
  - certainty; physical transcription; textual criticism,
  - names and dates; graphs and trees; figures and tables;
  - language corpora, manuscript description....

**The Chicago Pizza Model**
A useful metaphor for expressing modularity. To build a TEI pizza, take...
- the core modules
- whatever structural modules are needed
- the toppings of your choice
- your own modifications
  (and document them in an ODD)

**How does this model work?**
- Each module corresponds with a section of the main schema, within which
- declarations for each element are enclosed by a pattern, which can be redefined (to remove its contents)
- the status of patterns can be over-ridden in your own schema
- declarations for elements make heavy use of parameterised class system

**An example**
In a schema we write
```xml
include "tei.rnc" {
  p = element paragraph { content.p }
}
include "general.rnc"
include "figures.rnc"
include "linking.rnc" {
  ab = notAllowed
}
```
which includes two modules; does one renaming; and excludes one element.

**Element Classes**
- Most TEI elements are assigned to one or more
  - element classes, identifying their syntactic properties, or
  - attribute classes, identifying their attributes
- In the schema, each class is represented by a pattern
  - This provides a (relatively) simple way of
    - documenting and understanding the schema
    - modifying content models
    - facilitating customization
    - An alternative way of doing architectural forms

**Some TEI model classes**
- divn: structural elements like divisions (\(<div>, \<div>, \<div2>\)...) 
- divtop: elements which can appear at the start of a divn element (\(<head>, \<epigraph>, \<byLine>\)...) 
- chunk: paragraph-like elements (\(<sp>, \<p>, \<lg>\)...) 
- phrase: elements which appear within chunks (\(<hi>, \<foreign>, \<date>\)...)
**TEI attribute classes**

- **global**: attributes which are available to every element (n, lang, id, TEIform)
- **linking**: attributes for elements which have linking semantics (targType, targOrder, evaluate)

---

**The TEIFORM attribute**

Two main usages...

- protect applications from the effect of element renaming

  ```xml
  <titolo TEIform="title">...</titolo>
  ```

- protect applications from the effect of syntactic sugar

  ```xml
  <tag type="xyz">
  ```

can be rewritten as

  ```xml
  <xyz TEIform="tag">
  ```

---

**A case study: the Lampeter corpus**

See [http://www.tu-chemnitz.de/phil/english/real/lampeter/lamphome.htm](http://www.tu-chemnitz.de/phil/english/real/lampeter/lamphome.htm) (or look in the Oxford Text Archive)

- Fairly typical requirements for language corpora
  - light presentational tagging
  - structural markup for access
  - demographic information about text production
  - small number of tags to ease data capture and validation

- Implementation
  - modules: core modules, plus four additional modules
  - some extensions, many exclusions

---

**The Lampeter corpus view of the TEI**

```
include "tei.rnc"
include "general.rnc"
include "corpus.rnc"
include "figures.rnc"
include "transcr.rnc"
include "linking.rnc"
```
Three types of customization

1. Kill an element
\[ \text{ab} = \text{notAllowed} \]

2. Add a new element to a class
\[ \text{MyList} = \text{element MyList} \{ \text{attributes.class.global, [item]} \} \]

3. Rename an element
\[ p = \text{element parágrafo} \{ \text{content.p} \} \]

Possible practical answers

We may need to do some or all of:
- Define extensive additional modules, possibly containing much syntactic sugar, for new domains
- Suck in external DTDs, like MathML, SVG, and XHTML tables and forms (but we will need to address name clashes and universal namespace support may be a while coming)
- Use all and only those parts of the TEI we need to avoid tag overload for authors
- Add convenience attributes (eg to bypass purist XLink markup for URLs)

Possible practical answers

We may need to do some or all of:
- Define extensive additional modules, possibly containing much syntactic sugar, for new domains
- Suck in external DTDs, like MathML, SVG, and XHTML tables and forms (but we will need to address name clashes and universal namespace support may be a while coming)
- Use all and only those parts of the TEI we need to avoid tag overload for authors
- Add convenience attributes (eg to bypass purist XLink markup for URLs)

Why bother?

- The TEI is a well-known reference point
- Using the TEI enables
  - sharing of data and resources
  - shared modular software development
  - lower learning curve and reduced training costs
- The TEI is stable, rigorous, and well-documented
- The TEI is also flexible, customizable, and extensible in documented ways
- The architectural approach offers the best compromise for practical work.

The author vs the editor?

Hold on: do we need to use the same schema for authoring, editing, production, interchange, and archive? The TEI philosophy allows us to:

1. develop sample documents for a new domain using generic tools like <div> and type attributes
2. generate a private authoring DTD which uses domain-specific language:

```xml
<! - memorandum marked up in TEIMEMO -->
<memo>
  <front>
    <from>Ty Coon</from>
    <to>Ev Angelist</to>
    <date>Today</date>
  </front>
  <body>
    <div><p>Re your memorandum of <date>July 21st</date>, I think that the chance of us switching to XML in this company is minimal. See <xptr url="http://www.ourcompany.com/policy/"/></p></div>
</memo>
```